山东大学
SHANDONG UNIVERSITY

崇新学堂

2024 — 2025 学年第一学期

实验报告

课程名称：Introduction to EECS Lab

实验名称：Design Lab4 - Hitting the Wall

学生姓名：胡君安、陈焕斌、黄颢

实验时间：2024 年 10 月 24 日

1 Goals

The overall goal of this lab is to develop a signals and systems model of the brain/robot system and use it to understand its performance. We will achieve this in three stages:

- Develop a difference equation model of the wall finder system.
- Build a state machine model of the feedback system.
- Implement a robot brain that realizes the state machine controller.

2 Experimental step

Collect resources

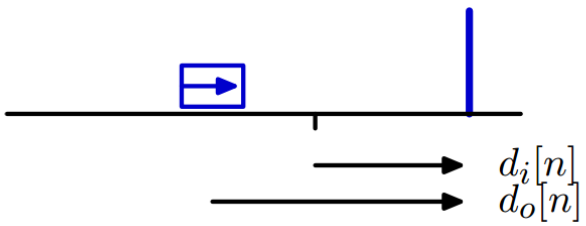
- **designLab04Work.py:** Template with appropriate imports for making state machines and plotting their outputs.
- **smBrainPlotDistSkeleton.py:**Template simple brain with a place for you to write the state machine for the controller.

This experiment aims to develop a simple model of a brain/robot system to control the distance between a robot and a wall. We define the following variables: the output distance $do[n]$ represents the current distance from the robot to the wall, the input distance $di[n]$ represents the desired distance, and the forward velocity v represents the robot' s constant forward speed. The time step T is set to 0.1 seconds, representing the interval between each command.

We assume that when the robot receives a new command, it immediately adjusts its speed and maintains this new velocity until the next command is received. The acceleration time of the robot is considered negligible.

During the experiment, the robot receives a command to adjust its distance from the wall and immediately changes its forward velocity v. This new velocity is held constant for the duration of the time step $T = 0.1$ seconds. This process repeats with each new command, continuously adjusting the robot' s position relative to the wall.

This model provides a simplified representation of a brain/robot system, focusing on the robot's immediate response and constant velocity maintenance between commands. The assumptions made allow us to ignore the acceleration time, simplifying the control dynamics of the robot.



Check Yourself 1

Given the following conditions, what is the distance to the wall on step 1?

$$v[0] = 1 \quad \mathrm{d_0}[0] = 3$$
$$v[1] = 2 \quad \mathrm{d_0}[1] = \boxed{2.9}$$

Check Yourself 2

Fully label the following system diagram. Include *do*, *di*, *ds*, *v*, *e*.

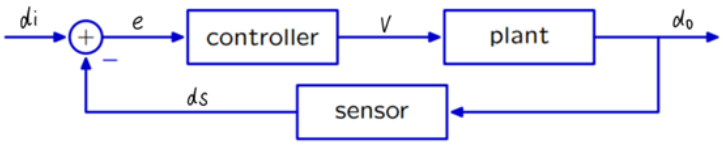


Fig. 1. The system diagram of Check Yourself 2

Determine difference equations (using constants T and k) to relate the input and output signals of the following system components.

- the controller : $\mathrm{v[n]} = \mathrm{ke[n]} = \mathrm{k(d_i[n] - d_s[n])}$
- the model of the plant : $\mathrm{d_0[n]} = \mathrm{d_0[n-1]} - \mathrm{T_v[n-1]}$
- the model of the sensor : $\mathrm{d_s[n]} = \mathrm{d_0[n-1]}$

Combine these equations to derive a difference equation that relates d_o to d_i, by:
Converting the difference equations to operator equations in R:

$$V = kE = k(D_i - D_s)$$
$$D_o = RD_o - RT_v$$
$$D_s = RD_o$$

Solving for D_o in terms of D_i:

$$D_o = RD_o - RTKE = RD_o - RTK(D_i - RD_o)(R^2Tk + R - 1)D_o = RTkD_iD_o = \frac{RTkD_i}{R^2Tk + R - 1}$$

Converting the result back to a difference equation:

$$d_o[n] = -Tkd_i[n-1] + d_o[n-1] + kTd_o[n-2]$$

Checkoff 1

"*Explain your results to a staff member.*"

This system has 3 parts: controller, plant and sensor. It is obvious that there is a feedback system in this WallFinder system. The controller sets velocity error which can be denoted by $e[n]$, the plant drives the robot to a new position depending on the given $v[n]$, and the sensor reports output but also introduces delay. We can successfully get the system diagram, from which we can get the difference equations. If we use R to represent the delay, we can successfully get the operator equations. Combine all operator equations, we are able to obtain the relation between D_o and D_i. Finally, we transform the R back to the delay so that we can convert the operator equation back to a difference equation.

Check Yourself 3

Use gains, delays, and adders to draw a system diagram for the first system in tutor problem Wk.4.2.1. The output at time n is the sum of its inputs up to and including time n.

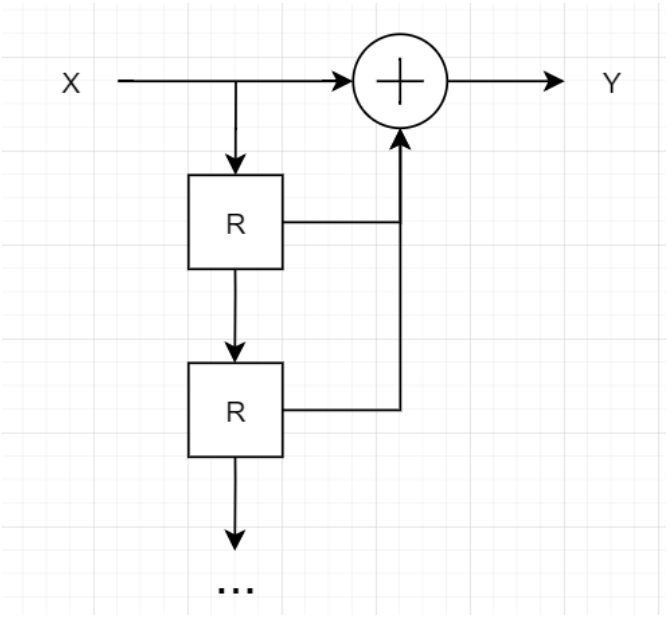


Fig. 2. The system diagram of Check Yourself 3

Check Yourself 4

Use gains, delays, and adders to draw a system diagram for the second system in tutor problem Wk.4.2.1.

The output at time n is the sum of its inputs up to and including time $n-1$.

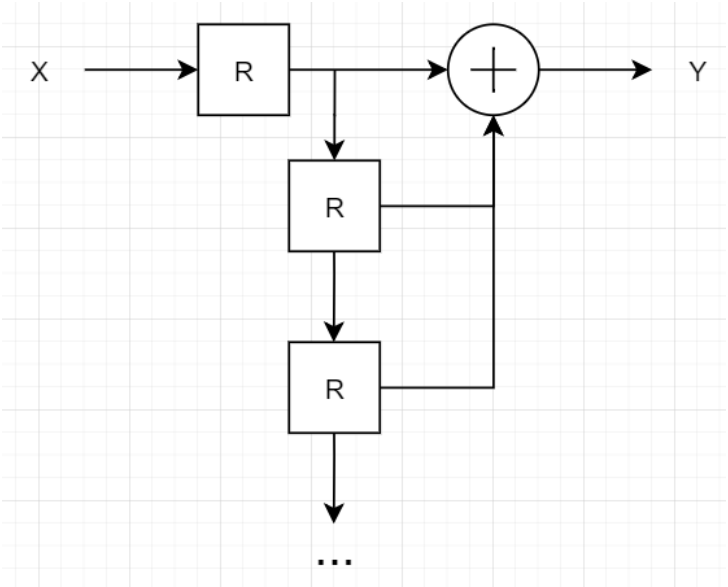


Fig. 3. The system diagram of Check Yourself 4

Check Yourself 5

Use gains, delays, and adders to draw a system diagram for the third system in tutor problem Wk.4.2.1.

The output at time n is the sum of the scaled inputs (each input scaled by 0.1) up to and including time $n-1$.

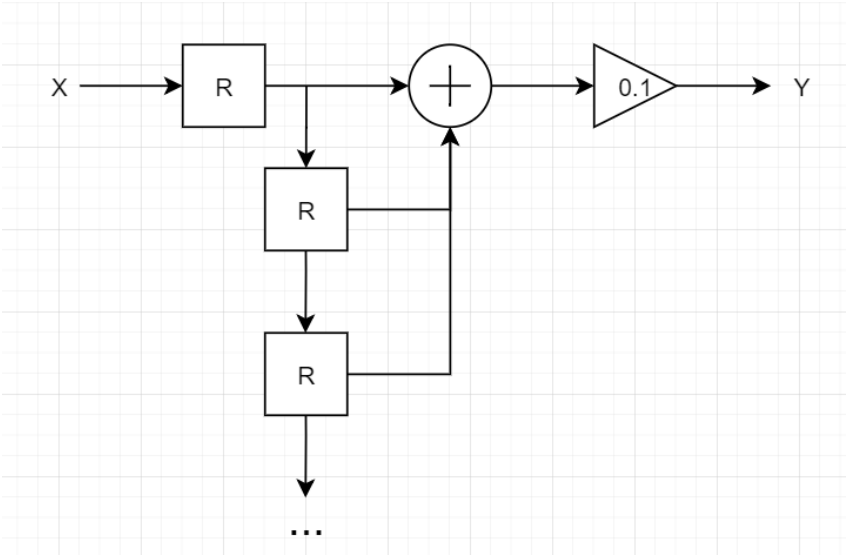


Fig. 4. The system diagram of Check Yourself 5

Wk.4.3.3

Do tutor problem Wk.4.3.3 (State machine composition).

```
def accumulator(init):
    return sm.FeedbackAdd(sm. Wire(), sm.R(init))

def accumulatorDelay(init):
    return sm.Cascade(sm.R(0), accumulator(init))

def accumulatorDelayScaled(s, init):
    return sm.Cascade(accumulatorDelay(init), sm.Gain(s))

if __name__ == '__main__':
    def test_accumulator():
        y = accumulator(0)
        print y.transduce( list( range(10)))

    def test_accumulatorDelay():
        y = accumulatorDelay(0)
        print y.transduce( list( range(10)))

    def test_accumulatorDelayScaled():
        y = accumulatorDelayScaled(0.5, 0)
        print y.transduce( list( range(10)))
```

```
22
23      test_accumulator()
24      test_accumulatorDelay()
25      test_accumulatorDelayScaled()
```

The output of the three functions is as follows:

```
1       IDLE 2.6.6
2       >>> ================================ RESTART ================================
3       >>>
4       [0, 1, 3, 6, 10, 15, 21, 28, 36, 45]
5       [0, 0, 1, 3, 6, 10, 15, 21, 28, 36]
6       [0.0, 0.0, 0.5, 1.5, 3.0, 5.0, 7.5, 10.5, 14.0, 18.0]
```

Check Yourself 6

Use gains, delays, and adders to draw a system diagram for the controller in the wall-finder system.

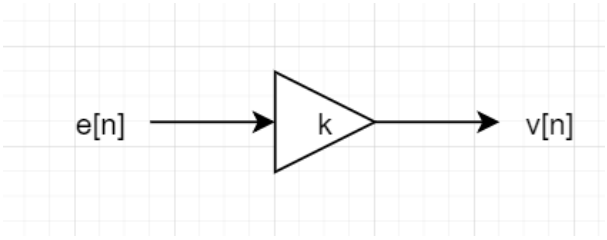


Fig. 5. The system diagram of Check Yourself 6

Check Yourself 7

Use gains, delays, and adders to draw a system diagram for the plant in the wall-finder system.

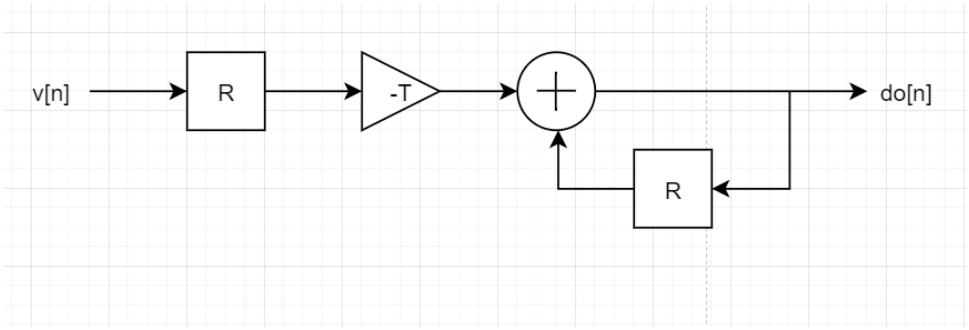


Fig. 6. The system diagram of Check Yourself 7

Check Yourself 8

Use gains, delays, and adders to draw a system diagram for the sensor in the wall-finder system.

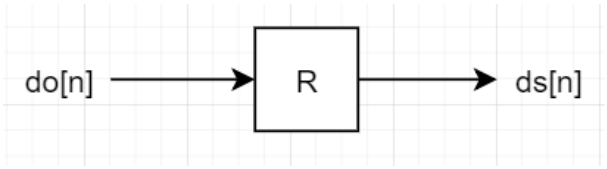


Fig. 7. The system diagram of Check Yourself 8

Check Yourself 9

Connect the previous three component systems to make a diagram of the wall-finder system. Label all the wires. Draw boxes around the controller ,plant, and sensor components.

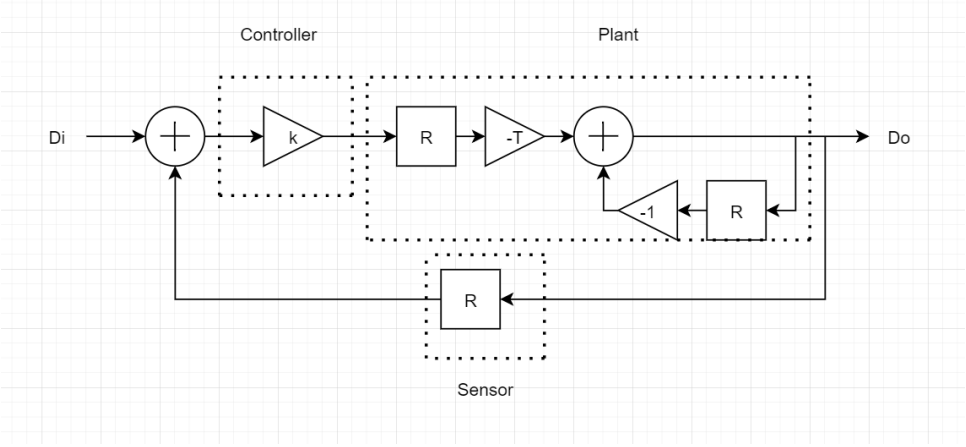


Fig. 8. The system diagram of Check Yourself 9

Checkoff 2

"Explain your system diagrams to a staff member. Identify instances of cascade and feedback composition for the wall-finder system."

Controller makes the $e[n]$ multiply k, to get $v[n]$. $v[n]$ goes into the Plant which gets $v[n]$ multiplied by $-T$, then add it and the delayed D_o (which also gets a negative number) to get D_o. D_o returns to the beginning through a feedback system. In this way, the robot can continuously get a suitable speed of its next action, which guarantees that it won' t hit a wall. In the wall-finder system, robot gets a signal of the distance to wall, it inputs the signal into the wall-finder system to get a corresponding speed which it needs. As speed changing, robot continue getting the signal of the distance to wall so that it can continuously get the speed it needs.

Wk.4.3.5

We have implemented a feedback control based state machine model for controlling the distance between robots and walls. It is divided into three main parts: controller, plant and sensor.We cascade the controller and plant through 'sm.FeedbackSubtract' , and then subtract the output of the sensor ,successfully integrated these three state machines into a closed-loop feedback control system – wallFinderSystem.

```
def plant(T, initD):
    return sm.Cascade(sm.Cascade(sm.R(0), sm.Gain(-T)),
                      sm.FeedbackAdd(sm. Wire(), sm.R(initD)))

def controller(k):
    return sm.Gain(k)

def sensor(initD):
    return sm.R(initD)

def wallFinderSystem(T, initD, k):
    return sm.FeedbackSubtract(sm.Cascade(controller(k), plant(T, initD)), sensor(initD))

initD = 1.5

def plotD(k, end = 50):
  d = ts.TransducedSignal(sig.ConstantSignal(0.7),
                          wallFinderSystem(0.1, initD, k))     # define the gain k
  d.plot(0, end, newWindow = 'Gain '+ str(k))

if __name__ == '__main__':
    plotD(1)
```

Check Yourself 10

Find three different values of k, one for which the distance converges monotonically, one for which it oscillates and converges, and one for which it oscillates and diverges. Make plots for each of these k values.

With T=0.1 and an initial distance to the wall of 1.5 meters, we experimented with different values of the gain – k and used PlotD (k, end) to plot the simulation result curve of distances to the wall. Here they are below.

When k=-1, the distance converges monotonically.

When k=-4, the distance oscillates and converges.

When k=1, the distance oscillates and diverges.

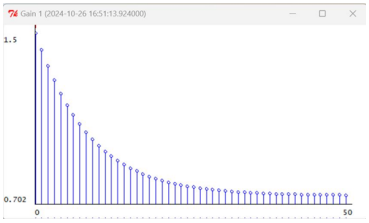


Fig. 9. k = -1

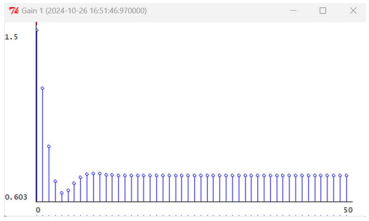


Fig. 10. k = -4

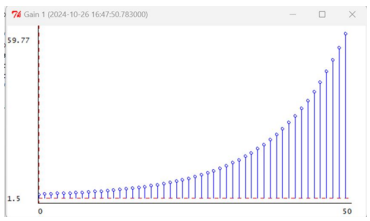


Fig. 11. k = 1

Wk.4.3.6: Wall Finder Gains

- The distance converges monotonically (without oscillation): -4 < k < 0

- The distance oscillates and converges (approaches a finite value): k < -4
- The distance oscillates and diverges (grows without bound): k > 0

Implement a brain for the wall-finder problem using a state machine

We have already implemented a Sensor state machine which outputs a delayed version of the values of sonar sensor 3.It takes as input the output of the sensor state machine, and generates as output instances of io.Action with 0 rotational velocity and an appropriate forward velocity. We edited the getNextValues method of the Controller class as below.We made the speed equal to the proportional coefficient k multiplied by the eps.

```
1       class Controller(sm.SM):
2       def __init__(self):
3           self.startState = 0
4           self.k = -3
5
6       def getNextValues(self, state, inp):
7           eps = dDesired - inp
8           v = self.k * eps
9
10          return state, io.Action(fvel=v, rvel=0)
```

Check Yourself 11

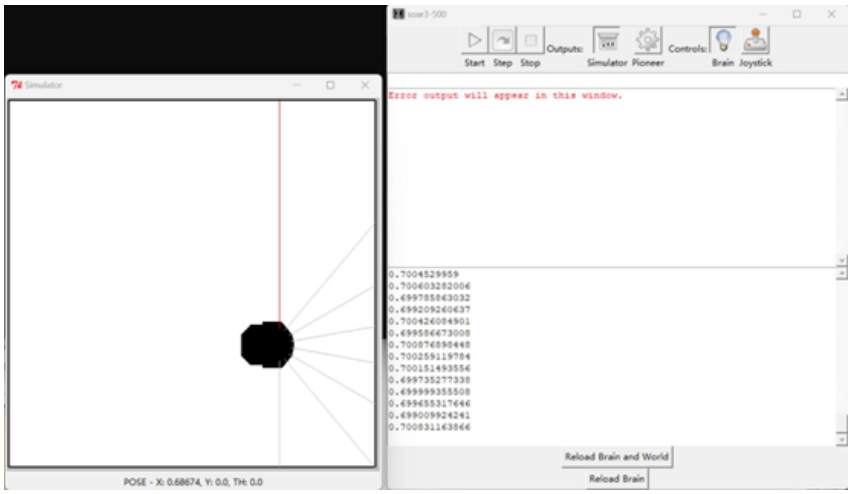


Fig. 12. Run the program in soar

For each of the three gains you found Check Yourself 10, run the simulated robot in the wallFinderWorld.py world, and save the plots.

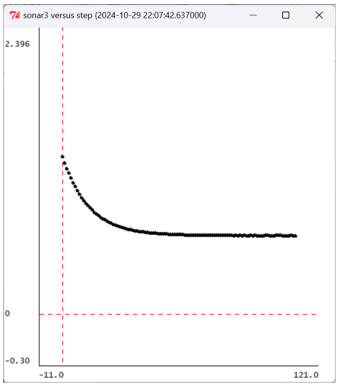


Fig. 13. k = -1

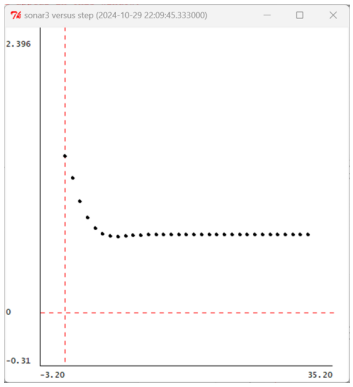


Fig. 14. k = -4

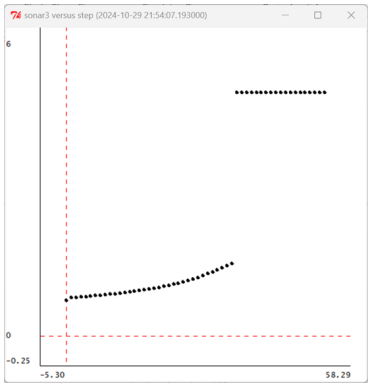


Fig. 15. k = 1

Checkoff 3

Compare the plots from Check Yourself 10 and 11. Explain how they differ, and speculate about why. Email your code and plots to your partner.

Through careful observation, we can find that the trend of the curve drawn in idle-n is basically the same as that drawn in soar, with only some abrupt points, due to the fast speed setting.

3 Summary

- After studying the wall-finder system, we successfully developed a difference equation model of the wall-finder system, built a state machine model of the feedback system, and implemented a robot brain that realizes the state machine controller.
- We successfully implemented a feedback control based state machine model for controlling the distance between robots and walls, and experimented with different values of the gain – k to observe the different behaviors of the system.
- When k=-1, the distance converges monotonically; when k=-4, the distance oscillates and converges; when k=1, the distance oscillates and diverges.